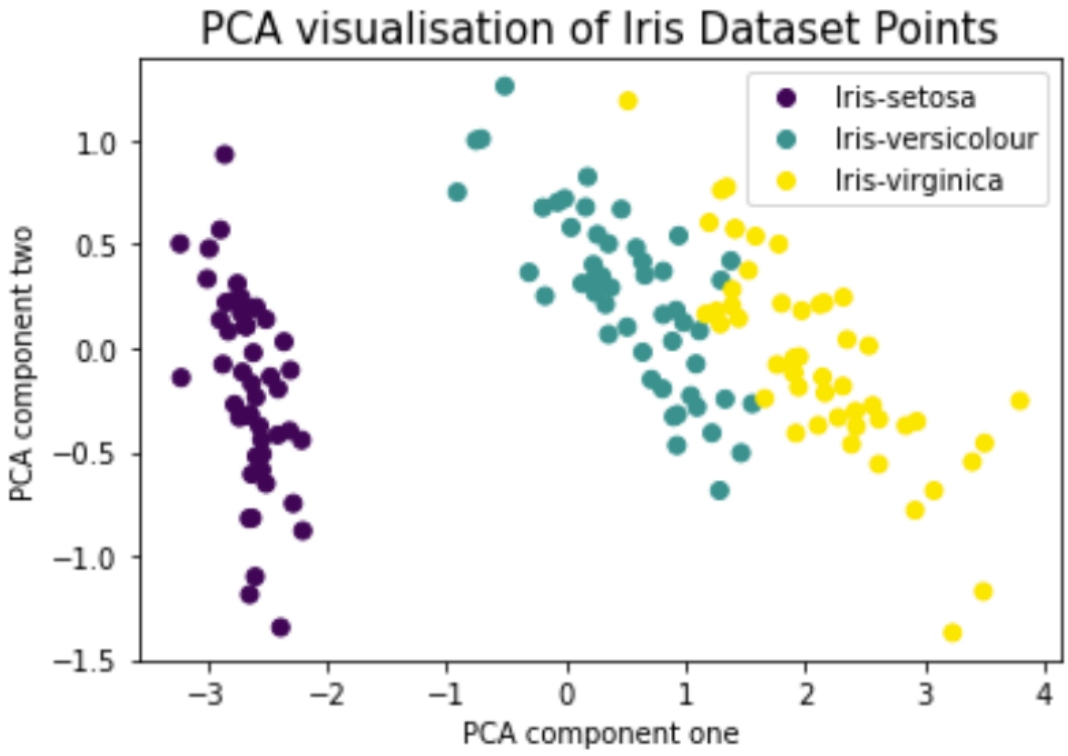
CS342 Machine Learning Coursework 2021

Student ID: 1907156
Word count: 883

This project uses the Iris Dataset from the UCI Machine Learning Repository [1]. It contains the sepal length, sepal width, petal length and petal width (in cm) of three different species of iris: Setosa, Versicolor and Virginica. Our model aims to classify iris species using the four measurements above. (all code included in .ipynb file).

1 Data visualization

Principal component analysis helps visualise data points in two dimensions by minimizing residual variance in the least-squares sense and maximizing the variance of the projection coordinates.



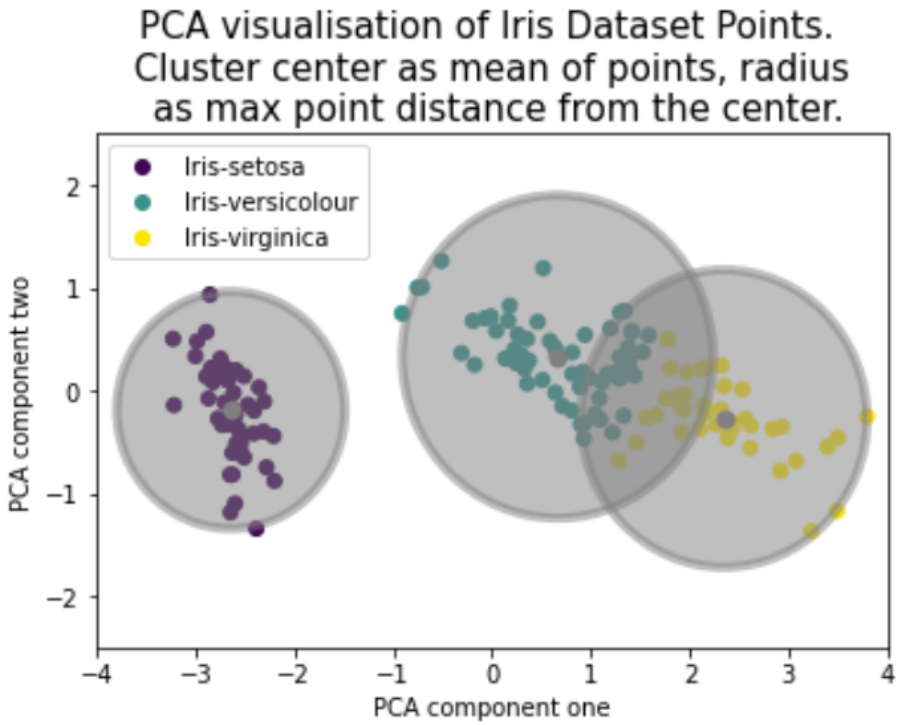
2 Clustering

A Gaussian Mixture Model is used on our data and assumes all data points are generated from finitely many Gaussian distributions, parameters: mean and variance (unknown).

Each iris species represents a cluster, where the initial means are used to predict our point labels and assign responsibility values as a base for the first iteration of the EM algorithm.

```
import sklearn.cluster as cl

# using K-means function to create the initial cluster means
kmeans = cl.KMeans(n_clusters = 3, random_state = 2).fit(transf_2)

# getting the cluster centers (so points) of our K-means
inital_means = kmeans.cluster_centers_
```

KMeans() assigns different labels to clusters every time it's run, we set the parameter "random_state=2" to stop the random assignment. The other option would be to run all permutations and compare them to the ground truth values.



Accuracy:

To quantitatively measure the percentage of correctly labelled points, we use the equation (Lab 3):

$$\text{Accuracy} = \frac{\text{No. of points in the correct cluster}}{\text{Total number of points}},$$

The initial labels prediction had an accuracy 0.887 (3 d.p.). We aim to improve the accuracy using the EM algorithm.

3 The Expectation-Maximization (EM) algorithm

Finds the maximum-likelihood estimates for model parameters. For simplicity, refer to the mean, probability and covariance matrix arrays as the MPCAs.

Parameters:
The set of points and initial responsibility values. (Calculated from the initial means, probabilities and covariances calculated at the start)

Method:

- Calculate new MPCAs of each cluster,
- Use updated MPCAs to find a new, better estimated r_c^i matrix,
- Compute convergence value and compares to the previous one,
- Repeat updating variables until the convergence value (initially decreases) starts increasing again,
- Then return the final variables (gives most optimal variables for label assignment).

Functions used:

	Calculates:	Parameters:	Return:
ric_calc()	Responsibility (r_c^i) of all cluster c for data-point x_i	MPCAs, and the set of points	Matrix of r_c^i values for every point for every cluster
mean_calc(), prob_calc(), cov_calc()	Mean, probability and covariance of clusters respectively	r_c^i matrix and the set of points	MPCAs respectively

Derived from specification equations (15), (18), (19) and (20) respectively.

Return: the best estimated r_c^i matrix, MPCAs (stored as final_ric, final_mean, final_prob, final_cov), and the number of iterations.

Implementation:

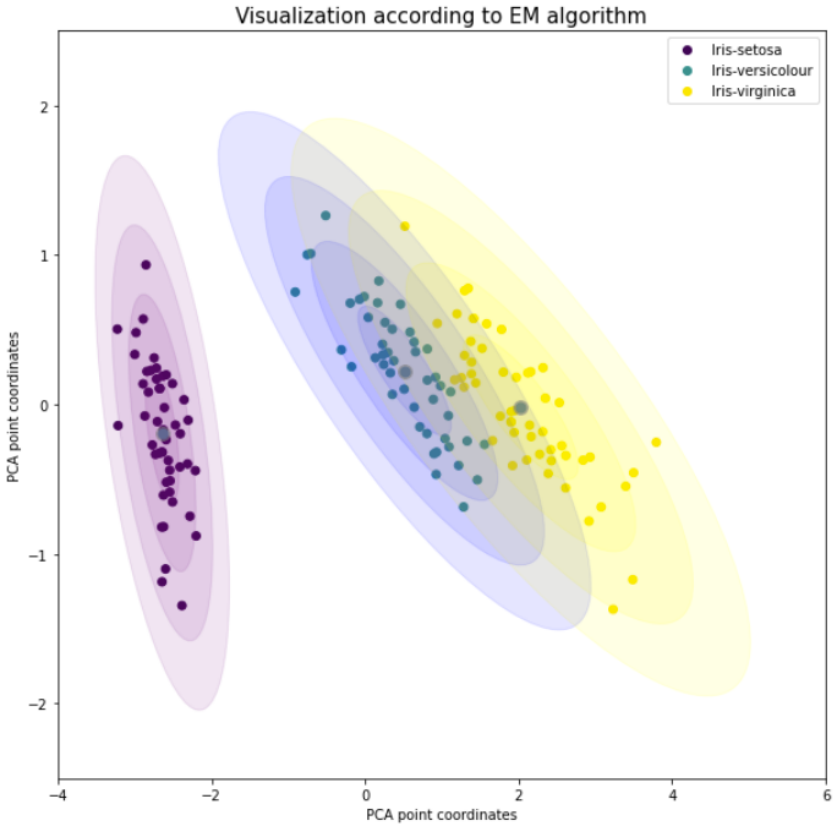
```
def EM(ric, points, threshold):
    mean = mean_calc(ric, points)
    prob = prob_calc(ric, points)
    cov = cov_calc(ric, mean, points)
    baseline = 1000

    # limits the iterations to 100 if the threshold is not reached
    for i in range(100):
        # stores the previous variables so we can compare the new ones but return 1
                                                    before
        prev_ric, prev_mean, prev_prob, prev_cov = ric, mean, prob, cov

        n_ric, n_pdfprob = ric_calc(mean,cov,prob,points)
        n_mean = mean_calc(n_ric, points)
        prob = prob_calc(n_ric, points)
        cov = cov_calc(n_ric, mean, points)

        new_baseline = conver(n_ric, n_pdfprob)

        # continue running to get a smaller convergence value
        # if the convergence starts increasing again, we break (first convergence
                                                    values does not count)

        if (i > 2) & (new_baseline < baseline):
            baseline = new_baseline
        elif (i > 2) & (new_baseline > baseline):
            print("Iterations: " + str(i))
            break

        # reassigning to new values
        mean = n_mean
        ric = n_ric

    return(prev_ric, prev_mean, prev_prob, prev_cov)
```

4 Testing

To obtain the initial responsibility values, we needed the initial means, probability and covariances (worked out previously), a necessity to find the probability distributions. The mathematical workings are derived from equation (4) of the specification.

Then the initial responsibly matrix is calculated in the same way as done in the ric_calc() method, but without using the built-in function mvn.pdf().

The accuracy function uses hard assignments (a point either belongs to the cluster or doesn't). Our final assignments are soft (a point has a probability of belonging to each cluster), therefore we convert them by assigning points to the highest probable cluster (highest responsibility). E.g. A point having r_c^i assignments of [0.2, 0.2, 0.8] would be given a value of 2 (assigned to cluster 3).



Now, the accuracy is 0.98 (with 20 iterations) which is greatly improved.

5 Convergence

The best measure is minimizing equation (16) from the specification, this is a rearrangement of the complete log-likelihood function. By minimising this, we maximize the marginal likelihood (the probability of how accurate the points are).

```
def conver(ric_mat, prob_mat):

    sumconver = 0
    for i in range(3):
        for j in range(len(ric[i])):
            # p(xi, zi|sample_space)
            prob_point = prob_mat[i][j]
            # ric value * log of
            sumconver += ric_mat[i][j]*math.log(prob_point)

    conver_val = (-1)*sumconver
    return(conver_val)
```

Initially, the convergence value decreases. We update the MPCAs and responsibility matrix and remeasure the value until it increases again, indicating our local minimum, as our convergence function is convex.

The first two values are incomparable because both are derived from the initial means prediction, rather than the EM algorithm.

Iterations: We ran 20 iterations to get the best result (150 data points). Why's this appropriate?:

- Had a very high accuracy of 0.98 (improvement from 0.833),
- Not ridiculously large, so running this on bigger datasets won't take too much time.

Alternative method: Comparing the difference between the new and previous values of the mean and r_c^i matrices can identify when the updated variable are converging, as the difference gets smaller (stop the iterations after a certain number to prevent infinite loops).

Although logical, it assumes a model slightly diverging away from the true model is better than before as only difference is compared, this is incorrect. Overall, it gives a lower accuracy (0.973 (3 d.p.)) than above and uses more iterations (52 iterations with threshold 0.01).

6 Discussion on accuracy

Our accuracy is based on the comparison of hard assigned values. This was suitable in calculating accuracy for the standard predicted values in section 2.

However, the EM method produces soft assigned values, requiring conversion into hard assignments to calculate accuracy.

Pros	Cons
Each point only has a single label (cluster),	Unless 100%, it may not belong to that cluster,
It's probabilistically valid and reasonable to assign the point to **its** cluster with highest probability.	A point with probability 0.51, 0.49 for clusters 1 and 2 respectively is assigned to cluster 1. However, there's a high probability it's in cluster 2.

Alternatives: The accuracy function could be altered to use soft assignments, by using the probability as a weighting.

E.g. take the point with soft assignment [0.6, 0.2, 0.2], belonging to cluster 2.

- original accuracy = $0/1 = 0$
- new accuracy = $0.2/1 = 0.2$

Problem: the point still indeed belongs to a single cluster, despite the probability of belonging to another.

Solution: randomly assign a cluster with a 0.6, 0.2 and 0.2 chance of being assigned to cluster 1, 2 and 3 respectively.

Limitations:
Probabilities were calculated using a multivariate normal distribution, we assume the data meets the following assumptions: [2]

- Residuals are normally distributed,
- There's a linear relationship between outcome and independent variables,
- Independent variables are not highly correlated with each other (E.g. sepal length and width), may not be the case.

7 References

[1] Fisher, R.A., Marshall, M. "Iris Data Set". `https://archive.ics.uci.edu/ml/datasets/Iris`. Published: 01.07.1988. Accessed: 19.11.2021.

[2] Statistic Solutions. "Assumptions of Multiple Linear Regression". `https://bit.ly/3HVImYD`. Accessed: 22.11.2021